

This Page Is Inserted by IFW Operations  
and is not a part of the Official Record

## **BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning documents *will not* correct images,  
please do not report the images to the  
Image Problem Mailbox.**

Title of the InventionA MEMORY ACCESS SYSTEMField of the Invention

The present invention relates to a memory access system particularly but not exclusively designed to facilitate memory accesses in a processor. The invention also relates to a method of generating memory addresses and for accessing memory, to a computer system and a computer program product.

Background to the Invention

Existing computer systems generally operate by generating memory addresses for accessing memory sequentially. That is the architecture of existing computer systems is arranged such that each memory access instruction defines a single memory address. Memory access units exist which allow two addresses to be generated from a single instruction, by automatically incrementing the address defined in the instruction by a certain predetermined amount. However, these systems are clearly restricted in that, if two addresses are generated, the second address necessarily bears a certain predetermined relationship to the first address. Vector stride units also exist which allow more than one memory address to be computed, but these are also limited in the relationship between the addresses. Moreover, it is necessary to generate the first address prior to calculating the second address and therefore it is not possible to generate two memory access addresses simultaneously in a single memory access unit.

It is an object of the present invention to provide increased flexibility for memory accesses.

Summary of the Invention

According to one aspect of the invention there is provided a memory access system comprising: a register file having a plurality of registers each having the same predetermined bit capacity and capable of holding one or more objects depending on the bit length of the object, said registers including at least a first address register holding at least two packed objects for use in identifying respective memory addresses; a register accessing circuit operative responsive to a memory access instruction to access said first address register in said register file; address generating circuitry for generating at least first and second memory addresses from said at least two packed objects; and memory access circuitry for accessing a memory using said first and second addresses.

In the embodiment described herein, the first address register is an index register, said at least two packed objects being offset objects. The second register holds the base value which, when combined with the packed objects, generates said first and second addresses.

A number of alternatives are possible. According to one alternative, the base value can itself comprise two packed objects which, when combined respectively with the two packed offset objects generate respective first and second addresses.

In another arrangement, the base value can comprise two packed objects, while the index register holds a single offset value which, when added to the respective packed base value objects generate said first and second memory addresses.

The address generating circuitry can comprise first and second addition circuits for respectively adding the packed objects in the first register with the contents of the second register to generate said first and second addresses.

The present invention also provides a computer system comprising: a memory holding data objects; a memory access unit for accessing said memory to retrieve data objects; a decode unit for decoding instructions for use in controlling said memory access unit; and a register file having a plurality of registers each having the

same predetermined bit capacity and capable of holding one or more objects depending on the bit length of the objects; said registers including at least a first address register holding at least two packed objects for use in identifying respective memory addresses; wherein the memory access unit is responsive to a memory access instruction defining said first address register to access said first address register in said register file and to generate at least first and second memory addresses from said at least two packed objects, said addresses being used to access said memory.

The length of the object to be retrieved from the memory can be defined in the memory access instruction.

The computer system can contain a second memory access unit which can itself generate two addresses from certain memory access instructions. However, if the second memory access unit is idle and two addresses have been generated by the first memory access unit, the second memory access unit can receive one of the addresses and use this so that two simultaneous accesses to memory can be made.

The invention also provides a method of generating addresses for accessing a data memory, the method comprising: retrieving from a base register a base value representing at least one base address; retrieving from an index register at least one index value; wherein at least one of the base value and the index value comprises at least two packed objects; combining the base value and the index value to generate at least first and second addresses for accessing said memory; wherein said base register and index register are defined in a single computer instruction.

Another aspect of the invention provides a method of accessing a memory holding data values, the method comprising: reading a memory access instruction which identifies at least a first address register holding at least two packed objects for use in identifying respective memory addresses; simultaneously generating at least first and second memory access addresses from said at least two packed objects; and using said first and second memory access addresses to access said memory.

A still further aspect of the invention provides a computer program product comprising program code means capable of cooperating with a computer when loaded therein to effect memory accesses, said program code means including a memory access instruction which identifies at least a first address register holding at least two packed objects for use in identifying respective memory addresses such that said first and second memory addresses are simultaneously generated by a computer on which the computer program product runs.

For a better understanding of the present invention and to show how the same may be carried into effect, reference will now be made by way of example to the accompanying drawings.

#### Brief Description of the Drawings

Figure 1 is a schematic block diagram illustrating a processor;  
Figure 2 is a diagram illustrating the encoding of two "packed" instructions;  
Figure 3 illustrates a number of different instruction formats;  
Figure 4 is a diagram illustrating operation of the memory access unit;  
Figures 5a and 5b illustrate the timing of different memory access operations;  
Figure 6 is a diagram of a 64 bit wide memory space;  
Figure 7 illustrates frame data; and  
Figure 8 illustrates frame data held in a 64 bit wide memory space.

---

#### Description of the Preferred Embodiment

The computer system described herein is a dual instruction stream, multiple data (DIMD) stream machine which normally executes two "instructions" every cycle. Figure 1 is a schematic diagram of the system. In Figure 1, reference numeral 2 denotes a program memory which holds programs in the form of a plurality of instructions. In the normal mode of operation, each 64 bit instruction in the program memory allows two 31 bit operations to be defined in the manner illustrated in Figure 2. That is, each 64 bit instruction contains two 31 bit instruction portions labelled

INST1 and INST2. Each instruction portion has associated with it a single bit which identifies the type of instruction. These are denoted ID1 and ID2 in Figure 2, standing for identification bits. An instruction portion can identify a data processing (DP) operation or a load/store (LD/ST) operation as will be explained more fully hereinafter. The allowed combinations are two data processing operations (ID1, ID2 both set to "0"), two load/store operations (ID1, ID2 both set to "1") or one data processing and one load store operation (ID1=0, ID2=1). Figure 3 illustrates some of the basic instruction formats for the 31 bit instruction portions, with M denoting the identification bit. In the following, the word "instruction" is used to denote the 32 bit sequence M plus the instruction portion because they are treated as instructions by each side of the DIMD machine.

Reverting to Figure 1, the program memory 2 is connected as instruction cache 3 which is connected to instruction fetch/decode circuitry 4. The fetch/decode circuitry issues addresses to the program memory and receives 64 bit lines from the program memory 2 (or cache 3), evaluates the opcode and transmits the respective instructions INST1, INST2 along X and Y channels  $5_x, 5_y$ . Each channel comprises a SIMD execution unit  $8_x, 8_y$  which includes three data processing units, MAC, INT and FPU and a load/store unit LSU 6. Each data processing unit MAC, INT and FPU and the load/store units LSU operate on a single instruction multiple data (SIMD) principle according to the SIMD lane expressed in the instruction according to the following protocol which defines the degree of packing of objects for packed data processing operations:

- (B) - 8 bit objects ( $b_0 \dots b_7$ )
- (H) - 16 bit objects ( $h_0 \dots h_3$ )
- (W) - 32 bit objects ( $w_0 \dots w_1$ )
- (L) - 64 bit objects (l)
- (S) - 32 bit floating point
- (D) - 64 bit floating point

For each channel  $5_x, 5_y$  if the instruction is a data processing instruction it is supplied

to the appropriate data processing unit MAC, INT or FPU and if it is a load/store instruction it is supplied to the load/store unit LSU. Data values are loaded to and from the MAC data processing units into and out of a common register file 10 which includes sixteen 64 bit special purpose registers along register access paths  $12_x, 12_y$ . Data values are loaded to and from the INT and FPU data processing units and the load/store units LSU into and out of a second register file 11 which includes sixty-four 64-bit general purpose registers. Register access paths  $14_x, 14_y$  are provided for these accesses. The program counter PC which indicates the current instruction can be read via one of the general purpose registers. Another one of the general purpose registers constitutes a control and status register.

Each register access path 12,14 carries three addresses from the accessing unit, two source addresses SRC1, SRC2 and a destination address DST. In the case of data processing instructions, the source addresses SRC1, SRC2 define registers in the register files 10, 11 which hold source operands for processing by the data processing unit. The destination address DST identifies a destination register into which a result of data processing will be placed. See instruction formats (1) to (5) in Figure 3. The operands and results are conveyed between the register file 10 or 11 and the respective data processing unit via the access paths 12, 14. In the case of load/store instructions, the instruction formats allow memory access addresses  $A_x, A_y$  to be formulated from data values held in the registers as described later. The load store units access a common address space in the form of a data memory 16 via a dual ported data cache DCACHE 15. For this purpose, each load/store unit has a 64 bit data bus  $D_x, D_y$  and a 64 bit address bus  $A_x, A_y$ . Each load/store unit  $6_x, 6_y$  can execute a number of different memory access (load/store) instructions, implemented in formats (6) and (7) in Figure 3.

According to a first class of memory access instructions, an object is loaded into a destination register specified in the instruction (in the DST field) from an address read from a source register in the instruction (in the BASE REG field). The length of the object depends on the SIMD lane B, H, W or L specified in the instruction opcode. If the object length is less than 64 bits, the upper bits of the destination register are

filled with zeros. This class of instruction also allows the number of objects to be specified. The memory address is read from the source register in the register file 11 by the specified load/store unit  $6_x, 6_y$  and despatched to the cache 15 via the appropriate address bus  $A_x, A_y$ . The object or objects are returned along the data bus  $D_x$  or  $D_y$  and loaded into the destination register of the register file 10 by the load/store unit. As is well known, on a cache miss, an access to the data memory 16 may be needed.

For each of the load instructions in the first class, there are matching store instructions. In each case, a single address is sent by each load/store unit and a single data value is returned at that address. That data value can constitute a number of objects depending on the number specified in the instruction and the length specified in the SIMD lane of the instruction.

The processor described herein additionally provides a new class of instructions which use packed offsets with a single base register in order to compute two (or more) addresses from a single instruction and therefore allow two (or more) data values to be retrieved from memory from one 32 bit instruction. The destination register (which serves as a source register for store operations) specifies an even/odd pair of registers for the memory access. Format (7) illustrated in Figure 3 is used for this class of memory access instructions. By way of example, one instruction in this class, LDL2, will be described.

---

The load instruction LDL2 allows two long words to be loaded into successive destination registers  $r_b, r_b+1$  from two independent addresses ADDR1, ADDR2 derived from a base address held in a register  $r_a$  identified in the BASE REG field and two packed offsets  $w0, w1$  held in a register  $r_c$  identified in the INDX REG field. The LDL2 instruction thus allows two independent addresses to be generated. The INDX OP field allows the degree of packing (SIMD lane) in the index register  $r_c$  to be defined.

Figure 4 illustrates operation of the LDL2 instruction. The instruction itself is illustrated at the top of Figure 4 identifying a destination register  $r_b$ , a base register



$r_a$  and an index register  $r_c$ . The opcode of the instruction defines the instruction as an LDL2 instruction and the INDX OP field identifies the packing of the index register  $r_c$ . In this case, the packing is two 32 bit words  $w0, w1$ . The value in the base register field is denoted  $V_a$  and constitutes the base address.

One load store unit which may be  $6_x$  or  $6_y$  is fully illustrated in Figure 4. It comprises a register access circuit 20 and buffers 22,24 for temporarily holding the values received from the base register  $r_a$  and index register  $r_c$ . In addition there are two addition circuits 26,28, and a memory access circuit  $30_x$ .

The register access circuit 20 allows the general purpose register file 11 to be accessed. When the instruction is received, the base register  $r_a$  and index register  $r_c$  are addressed and they return their respective values  $V_a, w0;w1$  to the temporary buffers 22,24 respectively. The first addition circuit 26 adds the value  $V_a$  to the first word  $w0$  in the index register  $r_c$  and the second addition circuit 28 adds the base value  $V_a$  to the second word  $w1$  in the index register  $r_c$ . In this way, two address ADDR1 and ADDR2 are generated to the cache 15. Then, the destination registers  $r_b, r_b+1$  are addressed by the register access circuit 20. In this case, the register address  $r_b$  addresses an even or odd pair of registers. The retrieved 64 bit objects OBJ1,OBJ2 from memory are thus supplied to the general purpose register file 11 in the destination registers  $r_b, r_b+1$  respectively.

---

An advantage of having packed offsets  $w0, w1$  in the index register  $r_c$  is that a single packed operation, e.g.  $addw$ , can simultaneously update both offsets (and thus both addresses) by the same amount.

Although only one of the load/store units is fully illustrated in Figure 4, it will be appreciated that the other load/store unit acts in the same way.

Each load/store unit 6 comprises a memory access circuit 30 which receives the addresses ADDR1,ADDR2 and can supply them to the cache 15 in successive memory access cycles as illustrated in Figure 5a. The arrangement of the processor

is such that the cache 15 has two access ports MP1,MP2. The X channel communicates via access port MP1. The Y channel communications via access port MP2. The memory access circuits  $30_x, 30_y$  of each of the X and Y channels can communicate with one another so that, if the Y channel memory access circuit is inactive, but two addresses have been generated from a single instruction by the X channel load/store unit  $6_x$ , then the memory access circuit of the Y channel  $30_y$  can receive the second address ADDR2 from the X channel so that both addresses ADDR1, ADDR2 can be supplied to the cache 15 in the same memory access cycle. This is illustrated in Figure 5b.

This can occur, for example, if of the two instructions fetched in each machine cycle, one instruction is a data processing instruction while the other instruction is a load/store instruction. In that case, one load/store unit may not be active while the other load/store unit is. By way of example, if an LDL2 instruction is received by the load/store unit LSU  $6_x$  in the X channel, two addresses ADDR1 and ADDR2 are generated. ADDR1 is fed to the address generator  $30_x$  while the X channel LSU  $6_x$  determines whether or not the Y channel load/store unit LSU  $6_y$  is busy. If the Y channel LSU  $6_y$  is not busy, the second address ADDR2 is conveyed to the access unit  $30_y$  of the Y channel load/store unit LSU  $6_y$ . Then, in one machine cycle, both ADDR1 and ADDR2 are issued to the data memory 16 at the respective ports MP1,MP2. If however the Y channel load/store unit LSU  $6_y$  is busy, the second address ADDR2 is retained in the X channel load/store unit  $6_x$  ready for dispatch to the data memory 16 on the successive machine cycle.

Although a single instruction, the LDL2 instruction, has been described above, it will be appreciated that a whole class of instructions is possible. In the first place, the length of the object retrieved from memory can be set in the instruction as with the first class of instructions. That is, 8 bit objects (B), 16 bit objects (H), 32 bit objects (W), or 64 bit objects (L) can be retrieved from memory. In addition, there are store instructions with the same semantics as the load instructions in each case. Thus, the set of instructions are named LDB2, LDH2, LDW2, LDL2, STB2, STH2, STW2, STL2.

In addition, the packing of the index register  $r_c$  can be set differently to allow different values to be added to the value  $V_a$  in the base register  $r_a$ . Thus, the index register could provide the following index options  $b_0, b_1$ ;  $b_2, b_3$ ;  $b_4, b_5$ ;  $b_6, b_7$ ;  $h_0, h_1$ ;  $h_2, h_3$ ;  $w_0, w_1$ .

As a further modification, the base register itself can be treated as packed. That is,  $V_a$  could comprise two packed objects  $W_0, W_1$  which, when combined respectively with the packed objects in the index register  $r_c$  generate the first and second addresses  $ADDR1$ ,  $ADDR2$ . This increases the independence between the first and second addresses, because the base values can be set a long distance from one another in this way.

According to a further modification, each instruction can be caused to generate more than two addresses. For example, if the index register holds four packed objects ( $b_0, b_1, b_2, b_3$ , or  $h_0, h_1, h_2, h_3$ ), then in principle four addresses can be generated by adding these objects respectively to the single base address  $V_a$  or the packed base address objects  $W_0, W_1$ . This would require additional addition units in the load/store units.

Another, conventional, load instruction LDDL allows two long words to be loaded from two adjacent addresses in the data memory 16 into successive destination registers from a base address in a source register SOURCE with an offset given in a source register  $SOURCE+1$ .

---

The provision of the LDDL and LDL2 instructions combined with the flexibility of cooperation of the load/store units make the computer system described herein particularly useful in an MPEG environment.

Figure 6 is a schematic diagram of a 64 bit wide memory showing by way of example the effect of the LDDL and LDL2 instructions. That is, the LDDL instruction withdraws operands from two adjacent addresses,  $ADDR1$  and  $ADDR_{nxt}$  while the LDL2 instruction recalls data from two wholly independent addresses,  $ADDR1$  and  $ADDR2$ . As has already been described, the addresses are determined from a base address

and respective offsets held in a register. In each instruction, the base address is held in a first source register. For the LDDL instruction, the offset giving ADDR1 is held in a subsequent source register and the next address ADDRnxt is generated automatically. The generation of the next address depends on the memory width and semantics.

Reference will now be made to Figures 7 and 8 to describe the particular usefulness of the LDDL and LDL2 instructions in an MPEG environment. Figure 7 illustrates frame data which comprises a number of frame lines FL each containing a plurality of pixels P. Q denotes the number of pixels per line, for example 720. Typically, data processing is carried out on individual blocks within the frame, of which an 8x8 block is denoted BLK in Figure 5. The top row of the block has pixels  $P_A$  to  $P_{A+7}$ , the next row  $P_{A+Q}$  to  $P_{A+Q+7}$ , etc. Figure 8 illustrates how frame data of the type illustrated in Figure 7 might be held in a 64 bit memory. Each pixel is represented by a single byte, and consequently 8 bytes (representing 8 pixels P) can be held in each addressable row of memory having a length of 64 bits. The memory is byte addressable, that is individual bytes can be addressed within the memory. However, each time a memory access is made by the LDL2 instruction, two 64 bit rows are retrieved. Consider for example that address [A] denotes the shaded byte in Figure 6 and represents the address of the first pixel  $P_A$  in the block of Figure 5. Consider that we are trying to access the pixels  $P_{A+Q}$  to  $P_{A+Q+7}$ , i.e. the second line of the 8x8 block BLK. The first pixel  $P_{A+Q}$  of that frame line is held at address [A+Q]. To access the row which contains the byte at address [A+Q] (denoted Row1 in Figure 8) a logical operation  $A+Q$  AND NOT 7 locates the first byte in Row1 (denoted by BYTE0, Row1) thus allowing that 64 bit object to be accessed by one word w0 in the packed index register  $r_c$ . The second word w1 in the index register  $r_a$  of the LDL2 instruction row to be simultaneously accessed and processed. Alternatively (although not strictly necessary for 8x8 blocks) an LDDL instruction could have been used to generate the first address and the address of the subsequent row. The address of the subsequent row, Rownxt can easily be generated by the logical operation  $A+Q+8$  AND NOT 7 which locates the first byte of the next row, labelled BYTE0, Rownxt and allows that object to be accessed. In this way the sequence of bytes representing the pixels  $P_{A+Q}$

to  $P_{A+Q+7}$  are accessed from the memory by the successive generation of two consecutive addresses.

The LDDL instruction is particularly useful for 16x16 blocks of pixels because it allows a complete line of 16 pixels (bytes) to be loaded from one instruction.

---